فرشته باقری - محمد امانلو

۸۱۰۱۰۰۰۸۴ - ۸۱۰۱۰۰۰۸۹

github repository: https://github.com/MohammadAmanlou/SWT-Fall03

last commit: 07ae80a651511c2893445b1764792ecb6219e84b

سوال اول

تفاوت اصلی آن‌ها در scope تست آنهاست. WebMvcTest برای تست لایه وب اپلیکیشن به کار می‌رود و مانند unit test برای این لایه عمل می‌کند. SpringBootTest برای تست کل context اپلیکیشن و ارتباط آن با وابستگی‌های خارجی مانند دیتابیس یا third party API استفاده می‌شود پس در واقع برای integration testing کل اپلیکیشن استفاده می‌شود.

ویژگی‌ها	WebMvcTest	SpringBootTest
تمرکز	لایه وب (کنترلرها، فیلترها)	تمام لایه ها و وابستگی ها
context برنامه	بخشی (فقط لایه وب)	کل کانتکست

• وابستگی های لود شده	فقط Beanهای مرتبط با وب مانند:
	`@Controller`
	`@RestController`
	`@ControllerAdvice`
		همه Beanهای برنامه

سرعت و اندازه	سبک و سریع‌تر	سنگین‌تر و کندتر
استفاده متداول	یونیت تست برای کنترلرها	تست یکپارچه تمام اجزای برنامه

سوال دوم

P = $(\neg a \wedge b) \vee (b \wedge c) \vee (\neg b \wedge \neg c)$

الف)

Number	a	b	c	p
1	T	T	T	T
2	T	T	F	F
3	T	F	T	F
4	T	F	F	T
5	F	T	T	T
6	F	T	F	T
7	F	F	T	F
8	F	F	F	T

ب)

Active clause: a -> b=T and c=False

Pairs: (2, 6)

Active clause: b -> c = T or (c = F and a=T)

Pairs: (1, 3), (1, 7), (2, 4), (3, 5), (5, 7)

Active clause: c -> b = F or (a = T and b = T)

pairs: (3, 4), (3, 8), (4, 7), (7, 8), (1, 2)

پ)

Active clause: a -> b=T and c=False

Pairs: (2, 6)

Active clause: b -> c = T or (c = F and a=T)

Pairs: (1, 3), (2, 4), (5, 7)

Active clause: c -> b = F or (a = T and b = T)

pairs: (3, 4), (7, 8), (1, 2)

اگر جفت های نوشته شده در این قسمت را بررسی کنیم می‌بینیم که اینها زیر مجموعه جفت های قسمت ب هستند چون در CACC ما آزادی عمل بیشتری داریم و می‌توانیم زوج هایی داشته باشیم که minor های آنها با هم متفاوت هستند ولی هر دو حالت predicate را کاور می‌کنند پس تعداد حالت هایی که برای کاور کردن CACC میتوانیم داشته باشیم بیشتر است.
البته می دانیم که اگر بتوانیم RACC را کاور کنیم، CACC را هم کاور کرده ایم.

ت)

داشتن clause coverage لزوما predicate coverage را تضمین نمی‌کند. در ادامه دو دسته بندی تست میدهیم که در یکی clause coverage باعث PC می‌شود و در دیگری نه.

1- CC and PC:

TFT

FTF

2- Just CC and no PC:

TTT

FFF

به صورت کلی CC نمیتواند PC را subsume کند.

سوال سوم

در این کد سه پارامتر ورودی داریم که بعد از partion کردن ورودی ها باید آنها را به صورت دو به دو مپ کنیم تا تاثیر آنها را ببینیم.

Price			discountRate			minPurchase		
P1	price < 0	invalid	D1	dR < 0	invalid	M1	mP < 0	invalid
P2	price = 0	invalid	D2	dR = 0	valid	M2	mP = 0	invalid
P3	price > 0	valid	D3	dR >0 and dR < 1	valid	M3	mP > 0	valid
			D4	dR = 1	valid			
			D5	dR > 1	invalid			

یک شرط نیز داریم:

price < minPurchase -> C1

price >= minPurchase -> C2

P1, D1, M1, C2	P2, D1, M2, C2	P3, D1, M3, C1
P1, D2, M2, C1	P2, D2, M3, C1	P3, D2, M1, C2
P1, D3, M3, C1	P2, D3, M1, C2	P3, D3, M2, C2

P1, D4, M1, C1	P2, D4, M2, C2	P3, D4, M3, C2
P1, D5, M1, C2	P2, D5, M2, C2	P3, D5, M3, C1

موارد زیر infeasible هستند:

P1, M3, C2	P2, M2, D1	P3, M2, C1
P1, M2, C2	P2, M1, D1	P3, M1, C1

موارد قرمز شده infeasible هستند.

تست‌ها:

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class DiscountCalculatorTest {

    @Test
    public void testP1_D1_M1_C2() {
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(-1, -0.1, 1));
    }

    @Test
    public void testP1_D2_M2_C2() {
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(-1, 0, 0));
    }

    @Test
    public void testP1_D3_M3_C1() {
```

```
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(-1, 0.5, 5));
    }

    @Test
    public void testP2_D1_M2_C1() {
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(0, -0.1, 0));
    }

    @Test
    public void testP2_D2_M3_C1() {
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(0, 0, 5));
    }

    @Test
    public void testP2_D3_M1_C2() {
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(0, 0.5, 1));
    }

    @Test
    public void testP3_D1_M3_C1() {
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(10, -0.1, 5));
    }

    @Test
    public void testP3_D2_M2_C2() {
        assertEquals("10.0",
DiscountCalculator.calculateDiscountedPrice(10, 0, 0));
    }

    @Test
    public void testP3_D3_M3_C2() {
        assertEquals("5.0",
DiscountCalculator.calculateDiscountedPrice(10, 0.5, 5));
    }
```

```
    @Test
    public void testP1_D4_M1_C1() {
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(-1, 1, -1));
    }

    @Test
    public void testP1_D5_M1_C2() {
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(-1, 1.1, 1));
    }

    @Test
    public void testP2_D4_M2_C2() {
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(0, 1, 0));
    }

    @Test
    public void testP2_D5_M2_C2() {
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(0, 1.1, 0));
    }

    @Test
    public void testP3_D4_M3_C2() {
        assertEquals("10.0",
DiscountCalculator.calculateDiscountedPrice(10, 1, 5));
    }

    @Test
    public void testP3_D5_M3_C1() {
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(10, 1.1, 5));
    }
}
```